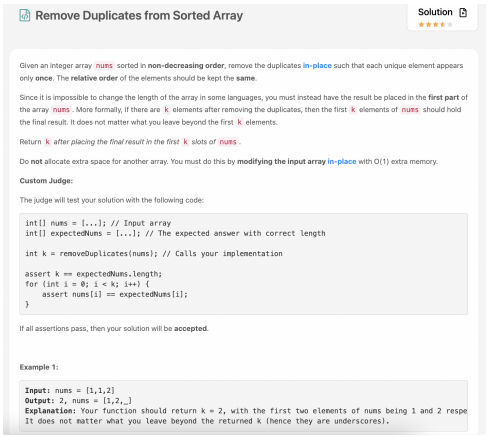
Arrays:

1. 

Remove Duplicates from Sorted Array

Solution

Given an integer array `nums` sorted in **non-decreasing order**, remove the duplicates **in-place** such that each unique element appears only **once**. The **relative order** of the elements should be kept the **same**.

Since it is impossible to change the length of the array in some languages, you must instead have the result be placed in the **first part** of the array `nums`. More formally, if there are `k` elements after removing the duplicates, then the first `k` elements of `nums` should hold the final result. It does not matter what you leave beyond the first `k` elements.

Return `k` *after placing the final result in the first* `k` *slots of* `nums`.

Do **not** allocate extra space for another array. You must do this by **modifying the input array in-place** with O(1) extra memory.

Custom Judge:

The judge will test your solution with the following code:

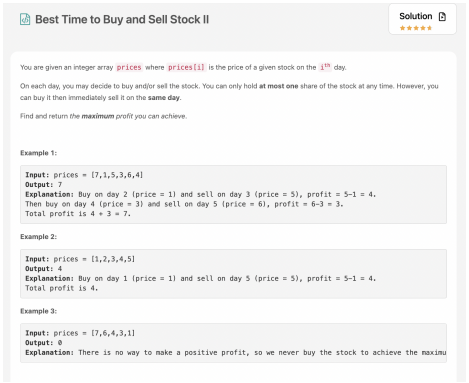
```
int[] nums = [...]; // Input array
int[] expectedNums = [...]; // The expected answer with correct length

int k = removeDuplicates(nums); // Calls your implementation

assert k == expectedNums.length;
for (int i = 0; i < k; i++) {
    assert nums[i] == expectedNums[i];
}
```

If all assertions pass, then your solution will be **accepted**.

Example 1:

```
Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respe
It does not matter what you leave beyond the returned k (hence they are underscores).
```

```
class Solution {
    public int removeDuplicates(int[] nums) {
        int size = nums.length;
        int index = 0;

        for(int i = 1; i < size; i++) {
            if(nums[i] != nums[index]) {
                nums[++index] = nums[i];
            }
        }
        return index + 1;
    }
}
```

2. 

Best Time to Buy and Sell Stock II

Solution

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the* ***maximum*** *profit you can achieve.*

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.
```

Example 2:

```
Input: prices = [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
Total profit is 4.
```

Example 3:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: There is no way to make a positive profit, so we never buy the stock to achieve the maximu
```

```
class Solution {
    public int maxProfit(int[] prices) {
        int profit = 0;
        for(int i = 1; i < prices.length; i++) {
            int diff = prices[i] - prices[i-1];
            if(diff > 0) {
                profit += diff;
            }
```

```
        }
        return profit;
    }
}
```

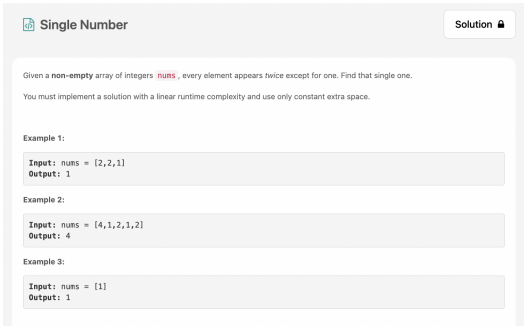
3. 


```
class Solution {
    public void rotate(int[] nums, int k) {
        if(k > nums.length)
            k=k%nums.length;

        int[] result = new int[nums.length];

        for(int i=0; i < k; i++) {
            result[i] = nums[nums.length-k+i];
        }
        int j=0;
        for(int i=k; i<nums.length; i++) {
            result[i] = nums[j];
            j++;
        }
        System.arraycopy( result, 0, nums, 0, nums.length );
    }
}
```

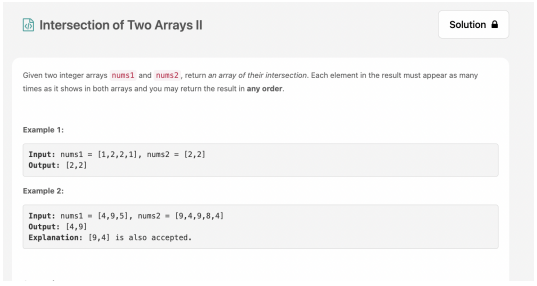
4. 


```
class Solution {
    public boolean containsDuplicate(int[] nums) {
        if((nums == null) || (nums.length==0)) {
```

```
            return false;
        }
        HashSet<Integer> set = new HashSet<Integer>();
        for(int i: nums) {
            if(!set.add(i)) {
                return true;
            }
        }
        return false;
    }
}
```

5. 

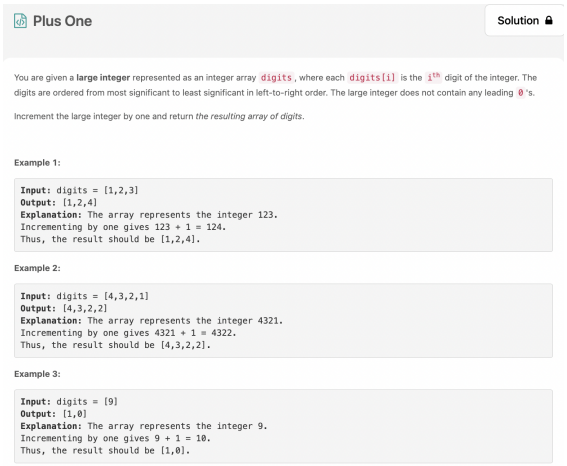

```
class Solution {
    public int singleNumber(int[] nums) {
        int result = 0;
        for (int x : nums) {
            result ^= x;
        }
        return result;
    }
}
```



6.
```
class Solution {
    public int[] intersect(int[] nums1, int[] nums2) {
        Arrays.sort(nums1);
        Arrays.sort(nums2);
```

```
        int sizeOfI = nums1.length;
        int sizeOfJ = nums2.length;

        if(sizeOfI > sizeOfJ){
            return intersect(nums2,nums1);
        }

        int i = 0, j = 0, k = 0;

        while((i < sizeOfI) && (j < sizeOfJ)) {
            if(nums1[i] < nums2[j]) {
                i++;
            } else if(nums1[i] > nums2[j]) {
                j++;
            } else{
                nums1[k++] = nums1[i];
                ++i;
                ++j;
            }
        }

        int result[] = new int[k];
        for(i = 0; i < k; i++){
            result[i] = nums1[i];
        }
        return result;
    }
}
```

7.



Plus One

Solution

You are given a **large integer** represented as an integer array `digits`, where each `digits[i]` is the i^{th} digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading `0`'s.

Increment the large integer by one and return *the resulting array of digits*.

Example 1:

```
Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].
```

Example 2:

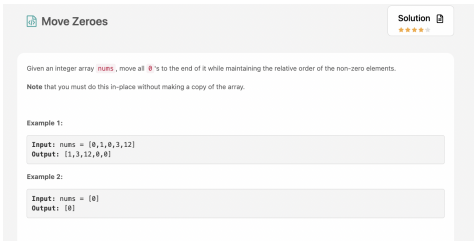
```
Input: digits = [4,3,2,1]
Output: [4,3,2,2]
Explanation: The array represents the integer 4321.
Incrementing by one gives 4321 + 1 = 4322.
Thus, the result should be [4,3,2,2].
```

Example 3:

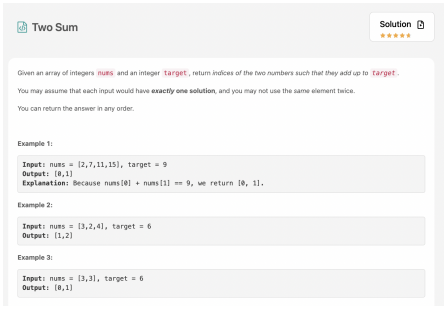
```
Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].
```

```
class Solution {
  public int[] plusOne(int[] digits) {
```

```
    int carry = 1;
    int n = digits.length;
    for (int i = n - 1; i >= 0; i--) {
      int temp = digits[i] + carry;
      if (temp <= 9) {
        digits[i] = temp;
        return digits;
      }
      digits[i] = temp % 10;
    }
    int[] newDigits = new int[n + 1];
    newDigits[0] = 1;
    for (int i = 1; i < n + 1; i++) {
      newDigits[i] = digits[i - 1];
    }
    return newDigits;
  }
}
```

8. 


```
class Solution {
    public void moveZeroes(int[] nums) {
        int element1 = 0;
        int element2 = 0;
        int size = nums.length;

        while(element1 < size) {
            if(nums[element1] != 0) {
                nums[element2] = nums[element1];
                element2++;
            }
            element1++;
        }
        while(element2 < size) {
            nums[element2++] = 0;
        }
    }
}
```

9. 

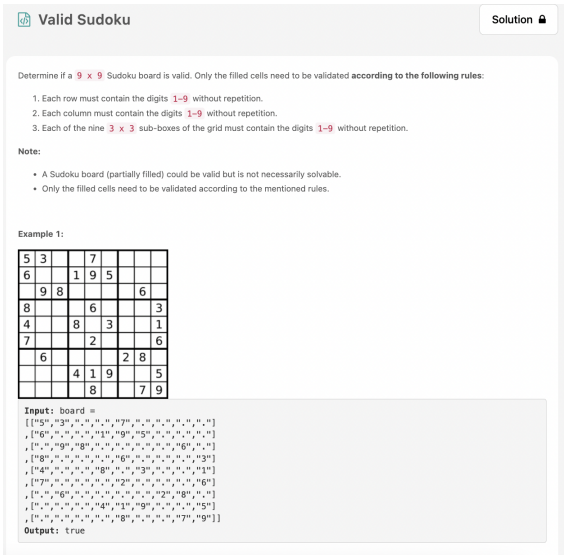
TC: O^2

```
public class Solution {
    public int[] twoSum(int[] nums, int target) {
        int[] result = new int[2];
        for (int i = 0; i < nums.length; i ++){
            for (int j = i + 1; j < nums.length; j++){
                if (nums[i] + nums[j] == target){
                    result[0] = i;
                    result[1] = j;
                    return result;
                }
            }
        }
        return result;
    }
}
```

Time complexity: O(n*log(n))

```
class TwoSum {
    private static int[] findTwoSum_Sorting(int[] nums, int target) {
        Arrays.sort(nums);
        int left = 0;
        int right = nums.length - 1;
        while(left < right) {
            if(nums[left] + nums[right] == target) {
                return new int[] {nums[left], nums[right]};
            } else if (nums[left] + nums[right] < target) {
                left++;
            } else {
                right--;
            }
        }
        return new int[] {};
```

```
    }
}
```

10. 

Valid Sudoku

Solution

Determine if a `9 x 9` Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the nine `3 x 3` sub-boxes of the grid must contain the digits `1-9` without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

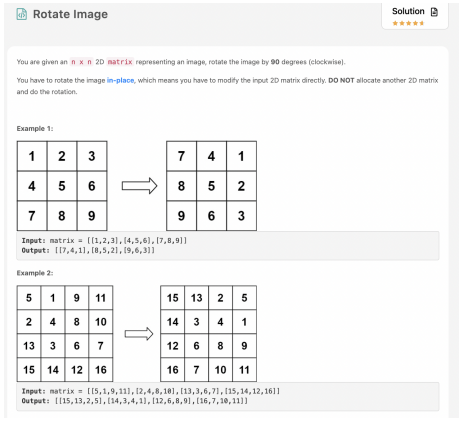
Example 1:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: true

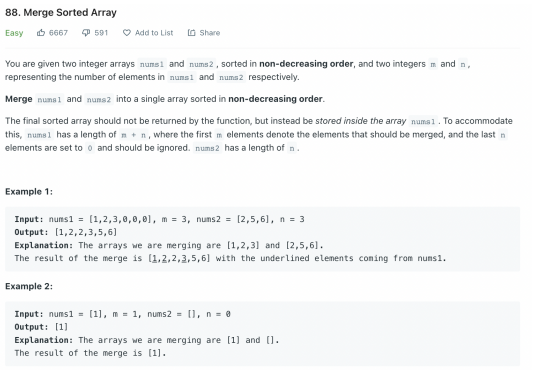
```
class Solution {
    public boolean isValidSudoku(char[][] board) {
        int n = board.length;
        Set<Integer>[] rows = new Set[n];
        Set<Integer>[] cols = new Set[n];
        Set<Integer>[] blocks = new Set[n];

        for(int i = 0; i < n; i++) {
            rows[i] = new HashSet<Integer>();
            cols[i] = new HashSet<Integer>();
            blocks[i] = new HashSet<Integer>();
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {

                if (board[i][j] == '.') continue;
                int k = 3 * (i /3) + (j / 3);

                int num = board[i][j] - '0';
                if (rows[i].contains(num)) return false;
                if (cols[j].contains(num)) return false;
                if (blocks[k].contains(num)) return false;

                rows[i].add(num);
                cols[j].add(num);
                blocks[k].add(num);
```

```
            }
        }
        return true;
    }
}
```

11. 


```
class Solution {
    public void rotate(int[][] matrix) {
        int n = matrix.length;
        for (int i = 0; i < n / 2; i++) {
            for (int j = 0; j < Math.ceil(((double) n) / 2.); j++) {
                int temp = matrix[i][j];
                                matrix[i][j] = matrix[n-1-j][i];
                                matrix[n-1-j][i] = matrix[n-1-i][n-1-j];
                                matrix[n-1-i][n-1-j] = matrix[j][n-1-i];
                                matrix[j][n-1-i] = temp;
                        }
                }
    }
}
```

12. Merge Sorted array(Two Pointers Approach)



88. Merge Sorted Array

Easy 6667 591 Add to List Share

You are given two integer arrays `nums1` and `nums2`, sorted in **non-decreasing order**, and two integers `m` and `n`, representing the number of elements in `nums1` and `nums2` respectively.

Merge `nums1` and `nums2` into a single array sorted in **non-decreasing order**.

The final sorted array should not be returned by the function, but instead be *stored inside the array* `nums1`. To accommodate this, `nums1` has a length of `m + n`, where the first `m` elements denote the elements that should be merged, and the last `n` elements are set to `0` and should be ignored. `nums2` has a length of `n`.

Example 1:

```
Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3
Output: [1,2,2,3,5,6]
Explanation: The arrays we are merging are [1,2,3] and [2,5,6].
The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.
```

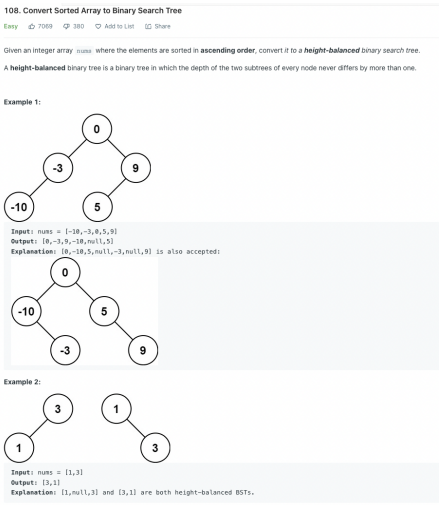
Example 2:

```
Input: nums1 = [1], m = 1, nums2 = [], n = 0
Output: [1]
Explanation: The arrays we are merging are [1] and [].
The result of the merge is [1].
```

TC: O(N + M). N

```
import java.util.Arrays;
class merge_sorted_array
{
   static void merge(int[] a , int m , int[] b , int n)
   {
      int i = m - 1 , j = n - 1 , idx = m + n - 1;
      while(i >= 0 && j >= 0)
      {
         if(a[i] >= b[j])
         {
            a[idx] = a[i];
            i--;
         }
         else
         {
            a[idx] = b[j];
            j--;
         }
         idx--;
      }
      while(i >= 0)
         a[idx--] = a[i--];
      while(j >= 0)
         a[idx--] = b[j--];
      return;
   }
   public static void main(String args[])
   {
      int m = 3 , n = 3;
      int[] a = new int[m + n];
      a[0] = 1;
      a[1] = 2;
      a[2] = 3;
```

```
        int[] b = {2 , 6 , 7};
        merge(a , m , b , n);
        for(int i = 0 ; i < a.length ; i++)
            System.out.print(a[i] + " ");
    }
}
```



108. Convert Sorted Array to Binary Search Tree

Easy 7069 380 Add to List Share

Given an integer array `nums` where the elements are sorted in **ascending order**, convert *it to a* ***height-balanced*** *binary search tree.*

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

Example 1:


```
Input: nums = [-10,-3,0,5,9]
Output: [0,-3,9,-10,null,5]
Explanation: [0,-10,5,null,-3,null,9] is also accepted:
```


Example 2:


```
Input: nums = [1,3]
Output: [3,1]
Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.
```

13.

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        if (nums.length == 0) {
                            return null;
        }

                    return sortedArrayToBST(nums, 0, nums.length - 1);
    }
    public TreeNode sortedArrayToBST(int[] nums, int start, int end) {
```

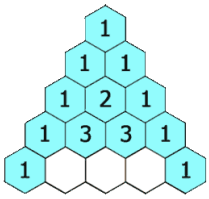
```
            if (start > end) {
                return null;
        }

            int mid = (start + end) / 2;
            TreeNode root = new TreeNode(nums[mid]);
            root.left = sortedArrayToBST(nums, start, mid - 1);
            root.right = sortedArrayToBST(nums, mid + 1, end);

            return root;
          }
}
```

118. Pascal's Triangle

Easy 7264 239 Add to List Share

Given an integer `numRows`, return the first numRows of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



Example 1:

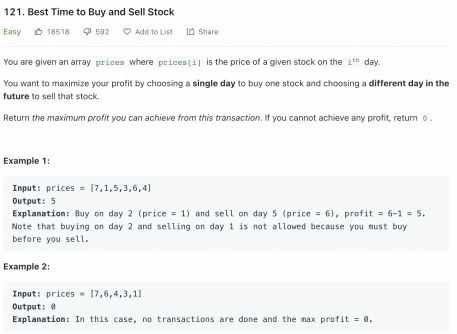
```
Input: numRows = 5
Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]
```

Example 2:

```
Input: numRows = 1
Output: [[1]]
```

14.

```
class Solution {
  public List<List<Integer>> generate(int numRows) {
    List<List<Integer>> result = new ArrayList<>();
    for (int i = 0; i < numRows; i++) {
      List<Integer> temp = new ArrayList<>();
      for (int j = 0; j <= i; j++) {
        temp.add(
          (j == 0 || j == i) ? 1 :
          (result.get(i - 1).get(j - 1) + result.get(i - 1).get(j)));
      }
      result.add(temp);
    }
    return result;
  }
}
```



121. Best Time to Buy and Sell Stock

Easy 18518 592 Add to List Share

You are given an array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

You want to maximize your profit by choosing a **single day** to buy one stock and choosing a **different day in the future** to sell that stock.

Return *the maximum profit you can achieve from this transaction*. If you cannot achieve any profit, return `0`.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 5
Explanation: Buy on day 2 (price = 1) and sell on day 5 (price = 6), profit = 6-1 = 5.
Note that buying on day 2 and selling on day 1 is not allowed because you must buy
before you sell.
```

Example 2:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: In this case, no transactions are done and the max profit = 0.
```

15.

```
import java.util.Scanner;
class BestTimetoBuyandSellStock {
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      // Prices array
      int prices[] = new int[]{7, 1, 5, 3, 6, 4};
      // Calculating the max profit
      int ans = maxProfit(prices, prices.length);
      // Print the answer
      System.out.println(ans);
   }
   private static int maxProfit(int[] prices, int n) {
      int maxSP[] = new int[n];
      int max = Integer.MIN_VALUE;
      // Construct the maxSP array
      for (int i = n - 1; i >= 0; i--) {
         if (prices[i] > max) {
            max = prices[i];
            maxSP[i] = Integer.MIN_VALUE;
         } else {
            maxSP[i] = max;
         }
      }
      int profit = 0;
      for (int i = 0; i < n; i++) {
         if (maxSP[i] != Integer.MIN_VALUE) {
            profit = Math.max(profit, maxSP[i] - prices[i]);
         }
      }
      // Return profit
      return profit;
   }
}
```

136. Single Number

Easy 10751 404 Add to List Share

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

```
Input: nums = [2,2,1]
Output: 1
```

Example 2:

```
Input: nums = [4,1,2,1,2]
Output: 4
```

Example 3:

```
Input: nums = [1]
Output: 1
```

16.

```
class Solution {
    public int singleNumber(int[] nums) {
        int result = 0;
        for (int x : nums) {
            result ^= x;
        }
        return result;
    }
}
```

17. Missing Number

268. Missing Number

Easy 6801 2946 Add to List Share

Given an array `nums` containing `n` distinct numbers in the range `[0, n]`, return *the only number in the range that is missing from the array.*

Example 1:

```
Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.
```

Example 2:

```
Input: nums = [0,1]
Output: 2
Explanation: n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.
```

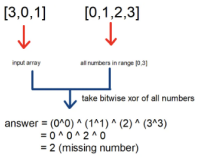
Example 3:

```
Input: nums = [9,6,4,2,3,5,7,0,1]
Output: 8
Explanation: n = 9 since there are 9 numbers, so all numbers are in the range [0,9]. 8 is the missing number in the range since it does not appear in nums.
```

Problem Statement

The **Missing Number LeetCode Solution** – "Missing Number" states that given an array of **size n** containing **n distinct numbers** between **[0,n]**. We need to return the number which is **missing** in the range.

Example:



answer = (0^0) ^ (1^1) ^ (2) ^ (3^3)
= 0 ^ 0 ^ 2 ^ 0
= 2 (missing number)

Input

Input: nums = [3,0,1]

Output

Output: 2

Explanation:

- We can easily observe that all the numbers between [0,3] are present except the number 2.
- Hence, 2 is the missing number in the range [0,3].

Input

Input: nums = [9,6,4,2,3,5,7,0,1]

Output

Output: 8

Explanation:

- All the numbers except 8, are present in the array for the range [0,9].
- Hence, 8 is the missing number in the range [0,9].

```
class Solution {
    public int missingNumber(int[] nums) {
        int n = nums.length,xo = n;
        for(int i=0;i<n;i++){
            xo^=i;
            xo^=nums[i];
        }
        return xo;
```

```
    }
}
```

18.

169. Majority Element

Easy 11025 366 Add to List Share

Given an array `nums` of size `n`, return *the majority element*.

The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Example 1:

```
Input: nums = [3,2,3]
Output: 3
```

Example 2:

```
Input: nums = [2,2,1,1,1,2,2]
Output: 2
```

```
import java.util.*;
class majority_element
{
   public static void main(String args[])
   {
      int[] a = {1 , 1 , 2 , 2 , 1 , 2 , 2};
      System.out.println(majorityElement(a));
   }
   static int majorityElement(int[] a)
   {
      int candidate = -1 , cnt = 0;
      for(int i = 0 ; i < a.length ; i++)
      {
         if(cnt == 0)
            candidate = a[i];
         cnt += (candidate == a[i]) ? 1 : -1;
      }
      return candidate;
   }
}
```

217. Contains Duplicate

Easy 5898 992 Add to List Share

Given an integer array `nums`, return `true` if any value appears **at least twice** in the array, and return `false` if every element is distinct.

Example 1:

```
Input: nums = [1,2,3,1]
Output: true
```

Example 2:

```
Input: nums = [1,2,3,4]
Output: false
```

Example 3:

```
Input: nums = [1,1,1,3,3,4,3,2,4,2]
Output: true
```

19.

```
class Solution {
    public boolean containsDuplicate(int[] nums) {
        if((nums == null) || (nums.length==0)) {
            return false;
        }
        HashSet<Integer> set = new HashSet<Integer>();
        for(int i: nums) {
            if(!set.add(i)) {
                return true;
            }
        }
        return false;
    }
}
```

20. Find Minimum Distance Between Two Numbers in an Array

Problem Statement

In the given unsorted array, which may also contain duplicates, find the minimum distance between two different numbers in an array.

Distance between 2 numbers in an array: the absolute difference between the indices +1.

Example

Input

12

3 5 4 2 6 5 6 6 5 4 8 3

3 6

Output

4

```
import static java.lang.Math.min;
import java.util.Scanner;
class sum
{
   public static int abs(int x)
   {
      if(x<0)
         x=x*-1;
      return x;
   }
   public static void main(String[] args)
   {
      Scanner sr = new Scanner(System.in);
      int n = sr.nextInt();
      int arr[] = new int[n];
      for(int i=0;i<n;i++)
      {
         arr[i] = sr.nextInt();
      }
      int X = sr.nextInt(); int Y = sr.nextInt(); //the elements between which minimum
distance is to be found
      int min_dist = 10000000;
      int i = 0, j = 0; //i and j to point at X and Y present somewhere in the array
      while(i < n && j < n)
      {
        if(arr[i] == X) //if we get X
          {
            while( j < n && arr[j] != Y) // we simply loop till we get Y
              j++;
            if(j < n && arr[j] == Y)
            min_dist = min(min_dist,abs(i-j));//we update the minimum distance if required
            i = j; // important step because as we got X,Y pair we can stand at present
position and loop forward for another pair
          }
        else if(arr[i] == Y)
        {
            while( j < n && arr[j] != X)
              j++;
            if(j < n && arr[j] == X)
            min_dist = min(min_dist,abs(i-j));
          i = j;
        }
        else
         i++;
```

```
        }
        System.out.println(min_dist);
    }
}
```

283. Move Zeroes

Easy 10217 259 Add to List Share

Given an integer array `nums`, move all `0`'s to the end of it while maintaining the relative order of the non-zero elements.

Note that you must do this in-place without making a copy of the array.

Example 1:

```
Input: nums = [0,1,0,3,12]
Output: [1,3,12,0,0]
```

Example 2:

```
Input: nums = [0]
Output: [0]
```

21.

```
class Solution {
    public void moveZeroes(int[] nums) {
        int n = nums.length;
        int i = 0, j = 0;
        while(j<n){
            if(nums[j]==0){
                j++;
            }
            else{
                int temp = nums[j];
                nums[j] = nums[i];
                nums[i] = temp;
                i++;
                j++;
            }
        }
    }
}
```

350. Intersection of Two Arrays II

Easy 4976 749 Add to List Share

Given two integer arrays `nums1` and `nums2`, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
Explanation: [9,4] is also accepted.
```

22.

```
import java.util.*;
class Rextester{

    public static int[] intersect(int[] nums1, int[] nums2)
    {
```

```
        Arrays.sort(nums1);
        Arrays.sort(nums2);

        int i=0,j=0,k=0;
        while(i<nums1.length && j<nums2.length)
        {
            if(nums1[i]<nums2[j]) i++;
            else if(nums1[i]>nums2[j]) j++;
            else{
                nums1[k++]=nums1[i];
                ++i;++j;
            }
        }

        int ans[]=new int[k];
        for(i=0;i<k;i++){
            ans[i]=nums1[i];
        }
        return ans;
    }

    public static void main(String args[])
    {
        int[] nums1={1,2,2,1};
        int[] nums2={2,2};
    int[] ans=intersect(nums1,nums2);
        for(int x:ans)
            System.out.print(x+" ");
    }
}
```

11. Container With Most Water

Medium 18895 1029 Add to List Share

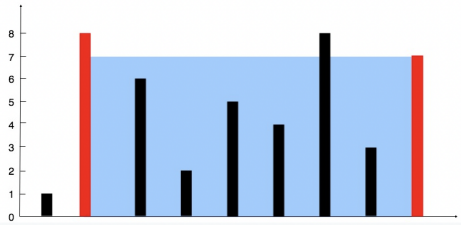
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the `i`th line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store*.

Notice that you may not slant the container.

Example 1:




```
Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.
```

Example 2:

```
Input: height = [1,1]
Output: 1
```

23.

```
class Solution {
    public int maxArea(int[] height) {
        int max = 0;

        int left = 0;
        int right = height.length - 1;

        while(left < right){
            int area = Math.min(height[left],height[right]) * (right - left);
            max = Math.max(max,area);

            if(height[left] < height[right]){
                left++;
            }else{
                right--;
            }
        }
        return max;
    }
}
```

15. 3Sum

Medium 19901 1886 Add to List Share

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j, i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does not matter.
```

Example 2:

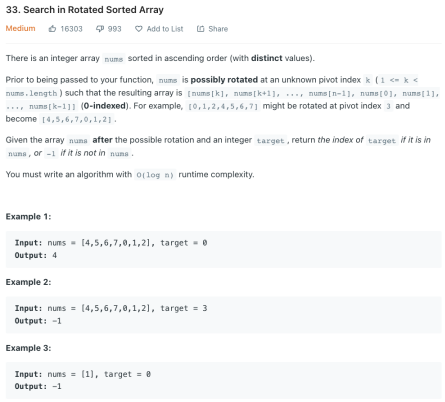
```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

24.

```
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
        List<List<Integer>> res = new ArrayList<>();
        Arrays.sort(nums);
        int target =0;

        for(int i=0; i<=nums.length-3 ; i++){
            // if num is same as previous, do nth
            if(i==0 || (i>0 && nums[i] != nums[i-1]) ){
                int sum = target - nums[i];
                int s = i+1 , e= nums.length-1;

                while(s<e){
                    int var_sum = nums[s] + nums[e];
                    if(var_sum == sum){
                        List<Integer> list = Arrays.asList(nums[i] , nums[s], nums[e]);
                        res.add(list);
                        // if duplicates present
                        while(s<e && nums[s]==nums[s+1]) s++;
                        while(s<e && nums[e]==nums[e-1]) e--;

                        s++;
                        e--;
                    }
                    else if (var_sum > sum)
                        e--;
                    else
                        s++;
                }
            }
        }
        return res;
    }
}
```



33. Search in Rotated Sorted Array

Medium 16303 993 Add to List Share

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, or `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

Example 2:

```
Input: nums = [4,5,6,7,0,1,2], target = 3
Output: -1
```

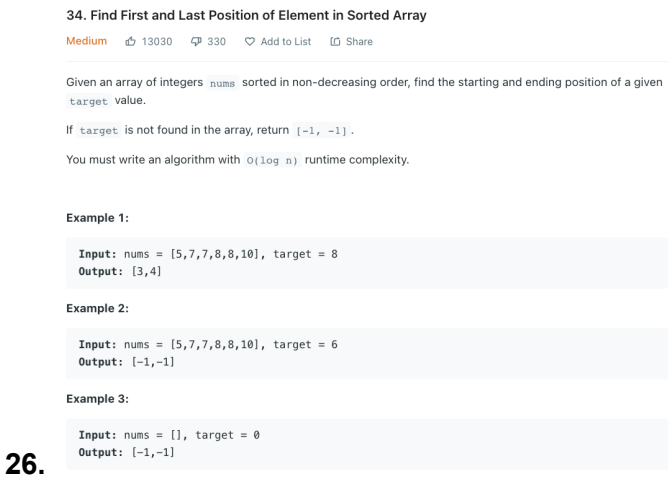
Example 3:

```
Input: nums = [1], target = 0
Output: -1
```

25.

```
import java.util.*;
import java.lang.*;
import java.io.*;
class Main {
    public static int search(int[] nums, int target) {
        int n = nums.length;
        int low = 0, high = n-1;
        while(low<=high){
            int mid = (low+high)/2;
            // check if the current element is target
            if(nums[mid] == target)
                return mid;
            // if the starting index of the search space has smaller element than current
element
            else if(nums[low]<=nums[mid]){
                // if target lies in non-rotated search space (or subarray)
                if(target >= nums[low] && target < nums[mid])
                    high = mid - 1;
                else
                    low = mid + 1;
            } else {
                // if target lies in non-rotated subarray
                if(target>nums[mid] && target<=nums[high])
                    low = mid + 1;
                else
                    high = mid - 1;
            }
        }
        // if you couldn't find the target element until now then it does not exists
        return -1;
    }

    public static void main(String[] args){
```

```
    int nums[] = {4,5,6,7,0,1,2};
    System.out.println(search(nums, 4));
  }
}
```



34. Find First and Last Position of Element in Sorted Array

Medium 13030 330 Add to List Share

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]
```

Example 2:

```
Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

Example 3:

```
Input: nums = [], target = 0
Output: [-1,-1]
```

26.

```
class Solution {
  public int first_pos(int nums[],int target){
    int start = 0,end = nums.length-1;

    while(start<=end){
      int mid = start + (end-start)/2;
      int midVal = nums[mid];

      if(midVal == target){
        if(mid == 0) return mid;

        if(midVal == nums[mid-1])
          end = mid-1;
        else
          return mid;
        }

      if(target > midVal)
        start = mid+1;
      else
        end = mid-1;
    }
  return -1;
  }
  public int last_pos(int nums[],int target){
    int start = 0,end = nums.length-1;
```

```
        while(start<=end){
            int mid = start + (end-start)/2;
            int midVal = nums[mid];

            if(midVal == target){
                if(mid == nums.length-1) return mid;

                if(midVal == nums[mid+1])
                    start = mid+1;
                else
                    return mid;
            }
            else if(target > midVal)
                start = mid+1;
            else
                end = mid-1;
        }
    return -1;
    }
    public int[] searchRange(int[] nums, int target) {
        int res [] = new int[2];
        Arrays.fill(res,-1);

        if(nums.length == 0)
            return res;

        res[0] = first_pos(nums,target);
        res[1] = last_pos(nums,target);
        return res;
    }
}
```

27.

46. Permutations

Medium 12037 207 Add to List Share

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Example 2:

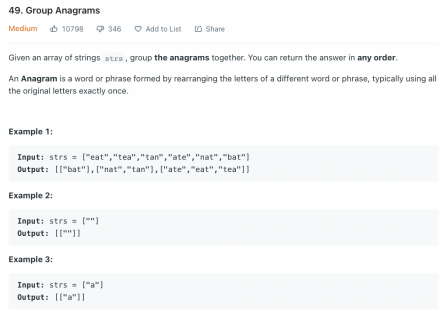
```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Example 3:

```
Input: nums = [1]
Output: [[1]]
```

```
class Solution {
    public List<List<Integer>> permute(int[] nums) {
    List<List<Integer>> result = new ArrayList<>();
```

```
    helper(0, nums, result);
    return result;
}
    private void helper(int start, int[] nums, List<List<Integer>> result){
        if(start==nums.length-1){
            ArrayList<Integer> list = new ArrayList<>();
            for(int num: nums){
                list.add(num);
            }
            result.add(list);
            return;
        }

        for(int i=start; i<nums.length; i++){
            swap(nums, i, start);
            helper(start+1, nums, result);
            swap(nums, i, start);
        }
    }

    private void swap(int[] nums, int i, int j){
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }
}
```



49. Group Anagrams

Medium 10798 346 Add to List Share

Given an array of strings `strs`, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

```
Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]
```

Example 2:

```
Input: strs = [""]
Output: [[""]]
```

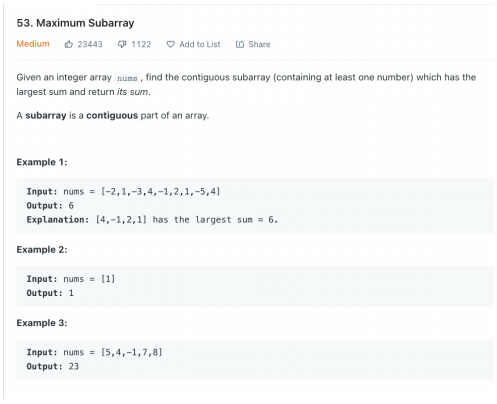
Example 3:

```
Input: strs = ["a"]
Output: [["a"]]
```

28.

```
public List<List<String>> groupAnagrams(String[] strs) {
    List<List<String>> result = new ArrayList<List<String>>();

    HashMap<String, ArrayList<String>> map = new HashMap<String,
ArrayList<String>>();
    for(String str: strs){
        char[] arr = new char[26];
        for(int i=0; i<str.length(); i++){
            arr[str.charAt(i)-'a']++;
```

```
        }
        String ns = new String(arr);

        if(map.containsKey(ns)){
            map.get(ns).add(str);
        }else{
            ArrayList<String> al = new ArrayList<String>();
            al.add(str);
            map.put(ns, al);
        }
    }

    result.addAll(map.values());

    return result;
}
```



53. Maximum Subarray

Medium 23443 1122 Add to List Share

Given an integer array `nums`, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

Example 1:

```
Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: [4,-1,2,1] has the largest sum = 6.
```

Example 2:

```
Input: nums = [1]
Output: 1
```

Example 3:

```
Input: nums = [5,4,-1,7,8]
Output: 23
```

29.

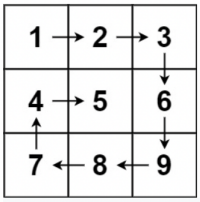
```
class Solution {
    public int maxSubArray(int[] nums) {
        int result = nums[0];
        int sum = nums[0];

        for(int i=1; i<nums.length; i++){
            sum = Math.max(nums[i], sum + nums[i]);
            result = Math.max(result, sum);
        }

        return result;
    }
}
```

54. Spiral Matrix

Medium 8216 885 Add to List Share

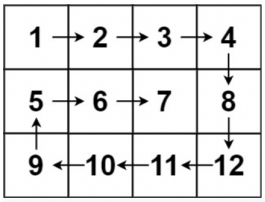
Given an `m x n` `matrix`, return *all elements of the* `matrix` *in spiral order*.

Example 1:




```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,2,3,6,9,8,7,4,5]
```

Example 2:




```
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

30.

```
class Solution {
   public List<Integer> spiralOrder(int[][] matrix)
   {
     int m=matrix.length;
       List<Integer>res=new ArrayList<>();
       if(m==0)
       return res;
      int n=matrix[0].length;

      int rowstart=0;
      int rowend=m;
      int colstart=0;
      int colend=n;

      int k;

      while(rowstart<rowend && colstart<colend)
      {

         for(k=colstart;k<colend;k++)
         res.add(matrix[rowstart][k]);
         rowstart+=1;

         for(k=rowstart;k<rowend;k++)
         res.add(matrix[k][colend-1]);
         colend-=1;
```

```
        if(rowstart<rowend)
        {
           for(k=colend-1;k>=colstart;k--)
           res.add(matrix[rowend-1][k]);
           rowend-=1;
        }

        if(colstart<colend)
        {
           for(k=rowend-1;k>=rowstart;k--)
           res.add(matrix[k][colstart]);
           colstart+=1;
        }

      }
      return res;
   }
}
```

55. Jump Game

Medium 12282 663 Add to List Share

You are given an integer array `nums`. You are initially positioned at the array's **first index**, and each element in the array represents your maximum jump length at that position.

Return `true` *if you can reach the last index, or* `false` *otherwise.*

Example 1:

```
Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.
```

Example 2:

```
Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is
0, which makes it impossible to reach the last index.
```

31.

```
class Solution {
   public boolean canJump(int[] nums) {
      int i = 0;
      for (int reach = 0; i < nums.length && i <= reach; ++i)
         reach = Math.max(i + nums[i], reach);
      return i == nums.length;
   }
}
```

32.

56. Merge Intervals

Medium 15193 555 Add to List Share

Given an array of `intervals` where `intervals[i] = [start_i, end_i]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

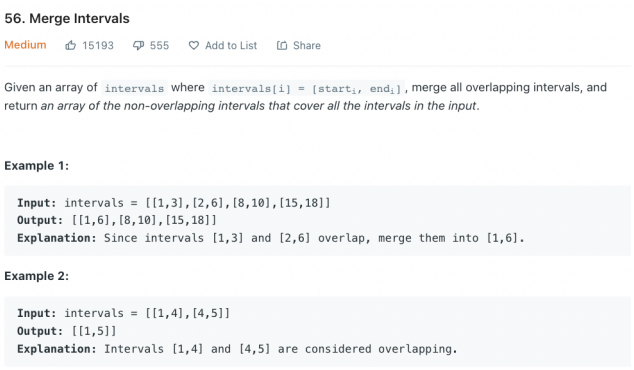
Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

```
public List<Interval> merge(List<Interval> intervals) {
    if(intervals == null || intervals.size()<=1){
        return intervals;
    }

    Collections.sort(intervals, Comparator.comparing((Interval itl)->itl.start));

    List<Interval> result = new ArrayList<>();
    Interval t = intervals.get(0);

    for(int i=1; i<intervals.size(); i++){
        Interval c = intervals.get(i);
        if(c.start <= t.end){
            t.end = Math.max(t.end, c.end);
        }else{
            result.add(t);
            t = c;
        }
    }

    result.add(t);

    return result;
}
```



56. Merge Intervals

Medium 15193 555 Add to List Share

Given an array of `intervals` where `intervals[i] = [start_i, end_i]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

33.

```
class Solution {
    public int[][] merge(int[][] intervals) {
   Arrays.sort(intervals, (a, b) -> {
     if (a[1] == b[1]) {
       return a[0] - b[0];
     }

     return a[1] - b[1];
   });

   LinkedList<int[]> list = new LinkedList<>();
   for (int[] interval : intervals) {
     if (!list.isEmpty() && list.getLast()[1] >= interval[0]) {

       while (!list.isEmpty() && list.getLast()[1] >= interval[0]) {
         interval[0] = Math.min(list.getLast()[0], interval[0]);
         interval[1] = Math.max(list.getLast()[1], interval[1]);
         list.removeLast();
       }
     }

     list.addLast(interval);
   }

   int pos = 0;
   int[][] answer = new int[list.size()][];
   for (int[] inteval : list) {
     answer[pos++] = inteval;
   }
   return answer;
    }
}
```

34.

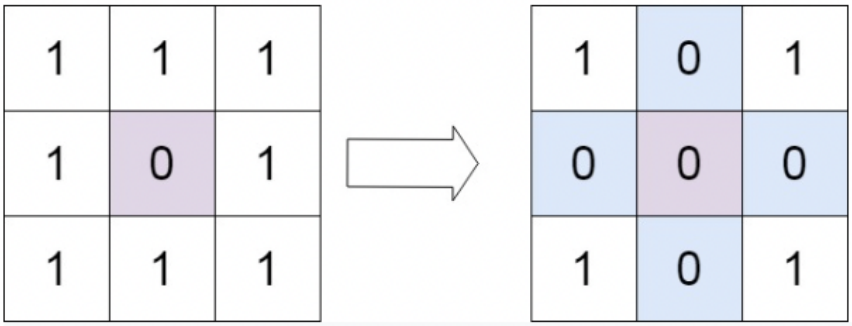
73. Set Matrix Zeroes

Medium 8396 528 Add to List Share

Given an `m x n` integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

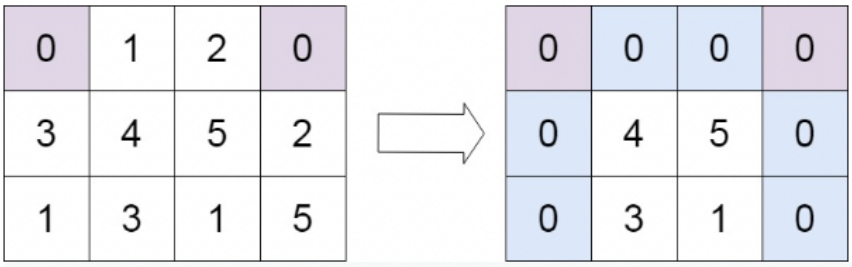
You must do it in place.

Example 1:




```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

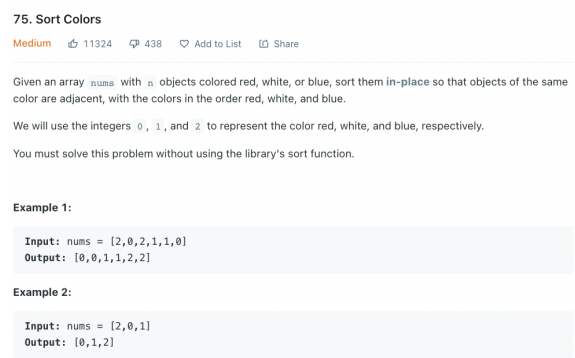
Example 2:




```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

```
class Solution {
    public void setZeroes(int[][] matrix) {
        boolean f1 = false,f2 = false;
        int n = matrix.length,m = matrix[0].length;
        for(int i=0;i<n;i++){
            if(matrix[i][0]==0){
                f1 = true;
            }
```

```
        }
        for(int j=0;j<m;j++){
            if(matrix[0][j]==0){
                f2 = true;
            }
        }
        for(int i=1;i<n;i++){
            for(int j=1;j<m;j++){
                if(matrix[i][j]==0){
                    matrix[i][0] = matrix[0][j] = 0;
                }
            }
        }
        for(int i=1;i<n;i++){
            for(int j=1;j<m;j++){
                if(matrix[i][0]==0 || matrix[0][j]==0){
                    matrix[i][j] = 0;
                }
            }
        }
        if(f1){
            for(int i=0;i<n;i++){
                matrix[i][0] = 0;
            }
        }
        if(f2){
            for(int j=0;j<m;j++){
                matrix[0][j] = 0;
            }
        }
    }
}
```



75. Sort Colors

Medium 11324 438 Add to List Share

Given an array `nums` with `n` objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

75. Sort Colors

Medium 11324 438 Add to List Share

Given an array `nums` with `n` objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

35.

```
public void sortColors(int[] nums) {
```

```
    if(nums==null||nums.length<2){
        return;
    }

    int[] countArray = new int[3];
    for(int i=0; i<nums.length; i++){
        countArray[nums[i]]++;
    }

    int j = 0;
    int k = 0;
    while(j<=2){
        if(countArray[j]!=0){
            nums[k++]=j;
            countArray[j] = countArray[j]-1;
        }else{
            j++;
        }
    }
}
```

36. Subsets

> 78. Subsets
>
> Medium 11276 167 Add to List Share
>
> Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.
>
> The solution set **must not** contain duplicate subsets. Return the solution in **any order**.
>
> **Example 1:**
>
> ```
> Input: nums = [1,2,3]
> Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
> ```
>
> **Example 2:**
>
> ```
> Input: nums = [0]
> Output: [[],[0]]
> ```

```
import java.util.*;
class Solution {
    List<List<Integer>> fl=new ArrayList<>();
```

```
    public void subSets(int[] nums, int p,List<Integer> l) {
      //if the pointer reaches beyond the last element in that case we have reached a valid
permutation and therefore it is added to the final list
          if(p==nums.length) {
          fl.add(new ArrayList<>(l));
          return;
          }
       l.add(nums[p]);//each element has 2 options either being added to the list or not
being added to the list
       subSets(nums,p+1,l);
       l.remove(Integer.valueOf(nums[p]));//not being added
       subSets(nums,p+1,l);//recursively calling to the next element
    }
    public List<List<Integer>> subsets(int[] nums)  {
       subSets(nums,0,new ArrayList<>());//starting from first element that is indexed at 0
       return fl;
    }
}
```

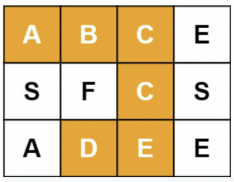
79. Word Search

Medium 10308 389 Add to List Share

Given an `m x n` grid of characters `board` and a string `word`, return `true` if `word` *exists in the grid.*

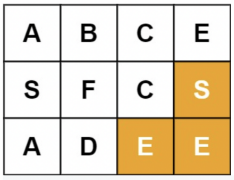
The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:



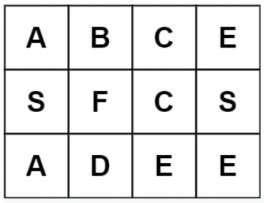

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
Output: true
```

Example 2:




```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"
Output: true
```

Example 3:




```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"
Output: false
```

37.

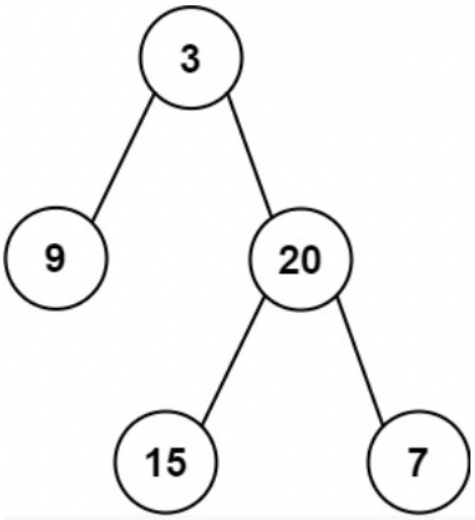
```
class Solution {
    public boolean exist(char[][] board, String word) {
        for(int i=0; i<board.length; i++){
            for(int j=0; j<board[0].length; j++){
                if(dfs(board, word, i, j, 0)){
                    return true;
                }
            }
```

```
        }

        return false;
    }

    public boolean dfs(char[][] board, String word, int i, int j, int k){
        if(board[i][j]!=word.charAt(k)){
            return false;
        }

        if(k>=word.length()-1){
            return true;
        }

        int[] di={-1,0,1,0};
        int[] dj={0,1,0,-1};

        char t = board[i][j];
        board[i][j]='#';

        for(int m=0; m<4; m++){
            int pi=i+di[m];
            int pj=j+dj[m];

if(pi>=0&&pi<board.length&&pj>=0&&pj<board[0].length&&board[pi][pj]==word.charAt(k
+1)){
                if(dfs(board,word,pi,pj,k+1)){
                    return true;
                }
            }
        }

        board[i][j]=t;

        return false;
    }
}
```

38. Construct Binary Tree from Preorder and Inorder Traversal

Given two integer arrays `preorder` and `inorder` where `preorder` is the preorder traversal of a binary tree and `inorder` is the inorder traversal of the same tree, construct and return *the binary tree*.

Example 1:




```
Input: preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]
Output: [3,9,20,null,null,15,7]
```

Example 2:

```
Input: preorder = [-1], inorder = [-1]
Output: [-1]
```

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public TreeNode buildTree(int[] preorder, int[] inorder) {
```

```
    int preStart = 0;
    int preEnd = preorder.length-1;
    int inStart = 0;
    int inEnd = inorder.length-1;

    return construct(preorder, preStart, preEnd, inorder, inStart, inEnd);
}

public TreeNode construct(int[] preorder, int preStart, int preEnd, int[] inorder, int inStart,
int inEnd){
    if(preStart>preEnd||inStart>inEnd){
        return null;
    }

    int val = preorder[preStart];
    TreeNode p = new TreeNode(val);

    //find parent element index from inorder
    int k=0;
    for(int i=0; i<inorder.length; i++){
        if(val == inorder[i]){
            k=i;
            break;
        }
    }

    p.left = construct(preorder, preStart+1, preStart+(k-inStart), inorder, inStart, k-1);
    p.right= construct(preorder, preStart+(k-inStart)+1, preEnd, inorder, k+1 , inEnd);

    return p;
}
}
```

39. Best Time to Buy and Sell Stock II

122. Best Time to Buy and Sell Stock II

Medium 8493 2422 Add to List Share

You are given an integer array `prices` where `prices[i]` is the price of a given stock on the i^{th} day.

On each day, you may decide to buy and/or sell the stock. You can only hold **at most one** share of the stock at any time. However, you can buy it then immediately sell it on the **same day**.

Find and return *the* ***maximum*** *profit you can achieve*.

Example 1:

```
Input: prices = [7,1,5,3,6,4]
Output: 7
Explanation: Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.
Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.
Total profit is 4 + 3 = 7.
```

Example 2:

```
Input: prices = [1,2,3,4,5]
Output: 4
Explanation: Buy on day 1 (price = 1) and sell on day 5 (price = 5), profit = 5-1 = 4.
Total profit is 4.
```

Example 3:

```
Input: prices = [7,6,4,3,1]
Output: 0
Explanation: There is no way to make a positive profit, so we never buy the stock to
achieve the maximum profit of 0.
```

```
class Solution {
    public int maxProfit(int[] prices) {
        int profit = 0;
        for(int i = 1; i < prices.length; i++) {
            int diff = prices[i] - prices[i-1];
            if(diff > 0) {
                profit += diff;
            }
        }
        return profit;
    }
}
```

40. Longest Consecutive Sequence

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in `O(n)` time.

Example 1:

```
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its
length is 4.
```

Example 2:

```
Input: nums = [0,3,7,2,5,8,4,6,0,1]
Output: 9
```

```
class Solution {
    public int longestConsecutive(int[] nums) {
        HashSet<Integer> set = new HashSet<>();
        for(int num: nums) set.add(num);

        int result = 0;

        for(int num: nums){
            int count = 1;

            int down = num-1;
            while(set.contains(down)){
                set.remove(down);
                down--;
                count++;
            }

            int up = num+1;
            while(set.contains(up)){
                set.remove(up);
                up++;
                count++;
            }

            result = Math.max(result, count);
        }
```

```
        return result;
    }
}
```

41.

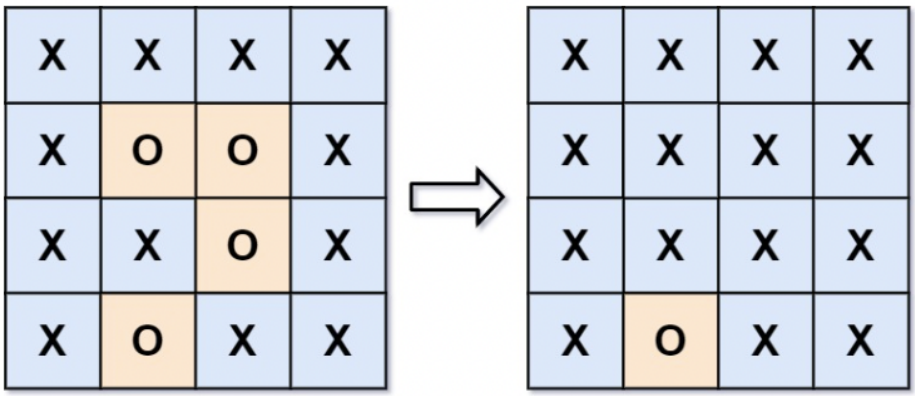
130. Surrounded Regions

Medium 5370 1289 Add to List Share

Given an `m x n` matrix `board` containing `'X'` and `'O'`, *capture all regions that are 4-directionally surrounded by* `'X'`.

A region is **captured** by flipping all `'O'`s into `'X'`s in that surrounded region.

Example 1:




```
Input: board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]
Output: [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]
Explanation: Notice that an 'O' should not be flipped if:
- It is on the border, or
- It is adjacent to an 'O' that should not be flipped.
The bottom 'O' is on the border, so it is not flipped.
The other three 'O' form a surrounded region, so they are flipped.
```

Example 2:

```
Input: board = [["X"]]
Output: [["X"]]
```

```
class Solution {
    public void solve(char[][] board) {
        if (board.length == 1 || board[0].length == 1) return;
        for (int i = 0; i < board[0].length; i++) {
            if (board[0][i] == 'O') visit(board, 0, i);
            if (board[board.length - 1][i] == 'O') visit(board, board.length - 1, i);
```

```
        }
        for (int i = 1; i < board.length - 1; i++) {
            if (board[i][0] == 'O') visit(board, i, 0);
            if (board[i][board[0].length - 1] == 'O') visit(board, i, board[0].length - 1);
        }
        for (int i = 0; i < board.length; i++) {
            for (int j = 0; j < board[0].length; j++) {
                if (board[i][j] == 'Y') board[i][j] = 'O';
                else board[i][j] = 'X';
            }
        }
    }

    private void visit(char[][] board, int i, int j) {
        if (i < 0 || i >= board.length || j < 0 || j >= board[0].length || board[i][j] != 'O') return;
        board[i][j] = 'Y';
        visit(board, i - 1, j);
        visit(board, i + 1, j);
        visit(board, i, j - 1);
        visit(board, i, j + 1);
    }
}
```

42. Maximizing Unique Pairs from two arrays

Given two arrays of equal size N, form maximum number of pairs by using their elements, one from the first array and second from the second array, such that an element from each array is used at-most-once and the absolute difference between the selected elements used for forming a pair is less than or equal to a given element K.

Examples:

```
Input : a[] = {3, 4, 5, 2, 1}
        b[] = {6, 5, 4, 7, 15}
          k = 3
Output : 4
The maximum number of pairs that can be formed
using the above 2 arrays is 4 and the corresponding
pairs are [1, 4], [2, 5], [3, 6], [4, 7], we can't
pair the remaining elements.
Other way of pairing under given constraint is
[2, 5], [3, 6], [4, 4], but count of pairs here
is 3 which is less than the result 4.
```

```
// Java implementation of above approach
import java.util.*;

class solution
{
```

```
// Returns count of maximum pairs that can
// be formed from a[] and b[] under given
// constraints.
static int findMaxPairs(int a[], int b[], int n, int k)
{
	Arrays.sort(a); // Sorting the first array.
	Arrays.sort(b); // Sorting the second array.

	// To keep track of visited elements of b[]
	boolean []flag = new boolean[n];
	Arrays.fill(flag,false);

	// For every element of a[], find a pair
	// for it and break as soon as a pair is
	// found.
	int result = 0;
	for (int i=0; i<n; i++)
	{
		for (int j=0; j<n; j++)
		{
			if (Math.abs(a[i]-b[j])<=k && flag[j]==false)
			{
				// Increasing the count if a pair is formed.
				result++;

				/* Making the corresponding flag array
				element as 1 indicating the element
				in the second array element has
				been used. */
				flag[j] = true;

				// We break the loop to make sure an
				// element of a[] is used only once.
				break;
			}
		}
	}
	return result;
}

// Driver code
public static void main(String args[])
{
```

```
        int[] a = {10, 15, 20};
        int[] b = {17, 12, 24};
        int n = a.length;
        int k = 3;
        System.out.println(findMaxPairs(a, b, n, k));

}
}
```

43.Maximum Gap

Leetcode – 164. Maximum Gap

Deepanshu Jain •

Given an integer array `nums`, return *the maximum difference between two successive elements in their sorted form*. If the array contains less than two elements, return `0`.

You must write an algorithm that **runs in linear time** and uses **linear extra space.**

Example 1:

```
Input: nums = [3,6,9,1]
Output: 3
Explanation: The sorted form of the array is [1,3,6,9], either (3,6) or (6,9) has the maximum difference 3.
```

Example 2:

```
Input: nums = [10]
Output: 0
Explanation: The array contains less than 2 elements, therefore return 0.
```

```
class Bucket {
  public int min;
  public int max;

  public Bucket(int min, int max) {
    this.min = min;
    this.max = max;
  }
}

class Solution {
  public int maximumGap(int[] nums) {
    if (nums.length < 2)
      return 0;

    final int min = Arrays.stream(nums).min().getAsInt();
    final int max = Arrays.stream(nums).max().getAsInt();
    if (min == max)
```

```
    return 0;

  final int gap = (int) Math.ceil((double) (max - min) / (nums.length - 1));
  final int bucketsLength = (max - min) / gap + 1;
  Bucket[] buckets = new Bucket[bucketsLength];

  for (int i = 0; i < buckets.length; ++i)
    buckets[i] = new Bucket(Integer.MAX_VALUE, Integer.MIN_VALUE);

  for (final int num : nums) {
    final int i = (num - min) / gap;
    buckets[i].min = Math.min(buckets[i].min, num);
    buckets[i].max = Math.max(buckets[i].max, num);
  }

  int ans = 0;
  int prevMax = min;

  for (final Bucket bucket : buckets) {
    if (bucket.min == Integer.MAX_VALUE) // empty bucket
      continue;
    ans = Math.max(ans, bucket.min - prevMax);
    prevMax = bucket.max;
  }

  return ans;
 }
}
```

44. Find Peak Element

162. Find Peak Element

Medium 6860 3902 Add to List Share

A peak element is an element that is strictly greater than its neighbors.

Given a **0-indexed** integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array.

You must write an algorithm that runs in `O(log n)` time.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.
```